Mikrokontroléry a embedded systémy

Hands-on: Mikrokontroléry STM32

Ing. Aleš POVALAČ, Ph.D.
povalac@vut.cz

Ústav radioelektroniky
Vysoké učení technické v Brně, FEKT

Školení pro SPŠ Třebíč

březen 2023

Seznámení s NUCLEO kity a STM32CubeIDE

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

- Založte nový projekt v STM32CubeIDE přes File / New / STM32 Project / Board Selector / NUCLEO-F030R8. Budeme využívat HAL knihovny, proto ponechte Targeted Project Type na STM32Cube. Potvrďte inicializaci všech periferií do výchozího nastavení.
- Překlad (Build all) lze spustit pomocí zkratky Ctrl+B, ladění (tj. spuštění, Debug) projektu pomocí F11.
- Při prvním spuštění se zobrazí výběr debuggeru, volte STM32 Cortex-M C/C++ Application. Pokud je firmware debuggeru v připojené vývojové desce neaktuální, může být vyžádán upgrade.
- Během krokování ladění lze s výhodou používat klávesové zkratky, viz menu Run.

- Git repo: https://github.com/alpov/SPST

Rozblikání LED a loopback
STM32F030

Rozblikání LED

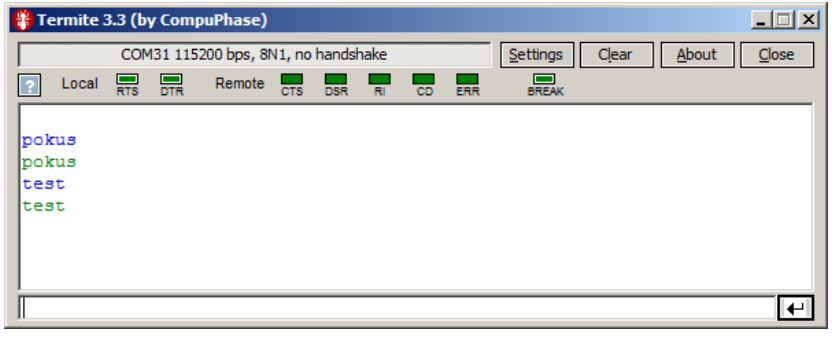
- Pin pro LD2 je ve výchozím stavu nastaven jako výstupní. Veškerou inicializaci zajišťuje kód vygenerovaný CubeMX.
  - Pro využití LED1 a LED2 na shieldu je třeba je nastavit (LED1=PA4, LED2=PB0)
- V souboru main.c doplňujte vlastní kód vždy mezi komentáře **USER CODE BEGIN** a **USER CODE END**. Pokud toto pravidlo porušíte, bude váš kód smazán při přegenerování projektu z CubeMX.
- Vývoj začněte funkcí main(). Do nekonečné smyčky vložte negaci příslušného pinu a krátké čekání.

```
HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
HAL_Delay(200);
```

UART loopback

- Zařízením loopback se rozumí okamžité odeslání každého přijatého bajtu zpět odesílateli. Pro testování sériové komunikace je optimální program Termite, výchozí rychlost portu je 38400 Bd.
- Funkci realizujeme pomocí HAL knihoven v nekonečné smyčce main()u:

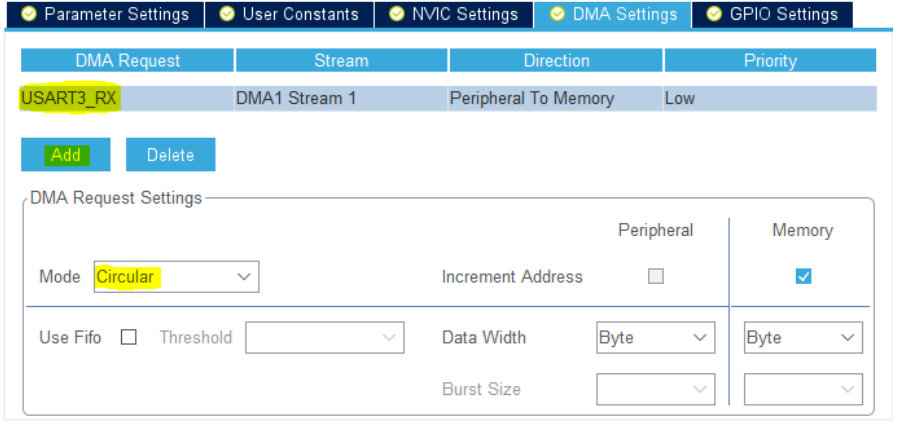
```
uint8_t c;
HAL_UART_Receive(&huart2, &c, 1, HAL_MAX_DELAY);
HAL_UART_Transmit(&huart2, &c, 1, HAL_MAX_DELAY);
```



DMA, obsluha textového protokolu STM32F429

Kruhový buffer s DMA (1/4)

- Pro rozhraní USART3 nastavte DMA Request pro příjem, využijte kruhový režim (circular).



- Příjem je nejvýhodnější realizovat s využitím kruhového bufferu. Jednotlivé přijímané znaky jsou postupně ukládány do bufferu, po dosažení jeho konce se automaticky pokračuje znovu od začátku. Zápisový index lze tedy vypočítat na základě délky bufferu a registrů DMA přenosu, čtecí index se obsluhuje softwarově.

Kruhový buffer s DMA (2/4)

- Deklarujeme buffer a indexy čtení a zápisu:

```
#define RX_BUFFER_LEN 64
static uint8_t uart_rx_buf[RX_BUFFER_LEN];
static volatile uint16_t uart_rx_read_ptr = 0;
#define uart_rx_write_ptr (RX_BUFFER_LEN - hdma_usart3_rx.Instance->NDTR)
```

- Nejdříve je třeba aktivovat DMA čtení z UARTu:

```
HAL_UART_Receive_DMA(&huart3, uart_rx_buf, RX_BUFFER_LEN);
```

- V hlavní smyčce kódu pak postupně zpracováváme jednotlivé bajty z bufferu. Dojde-li k zablokování kódu po určitou dobu (např. vlivem zpracování ISR), data se „nahromadí“ v bufferu a zpracují se, jakmile je k dispozici procesorový čas:

```
while (uart_rx_read_ptr != uart_rx_write_ptr) {
    uint8_t b = uart_rx_buf[uart_rx_read_ptr];
    if (++uart_rx_read_ptr >= RX_BUFFER_LEN) uart_rx_read_ptr = 0;
                                                    // increase read pointer

    uart_byte_available(b); // process received byte with the RX state machine
}
```

Kruhový buffer s DMA (3/4)

- Funkce **uart_byte_available()** ukládá tisknutelné znaky do dalšího bufferu, který již obsahuje poskládaný textový příkaz. Po nalezení konce řádku zavolá obsluhu pro vyhodnocení příkazu:

```
static void uart_byte_available(uint8_t c)
{
    static uint16_t cnt;
    static char data[CMD_BUFFER_LEN];

    if (cnt < CMD_BUFFER_LEN) data[cnt++] = c;
    if (c == '\n' || c == '\r') {
        data[cnt - 1] = '\0';
        uart_process_command(data);
        cnt = 0;
    }
}
```

- Vhodnou velikost bufferu pro textový příkaz **CMD_BUFFER_LEN** je třeba definovat (např. 256B).

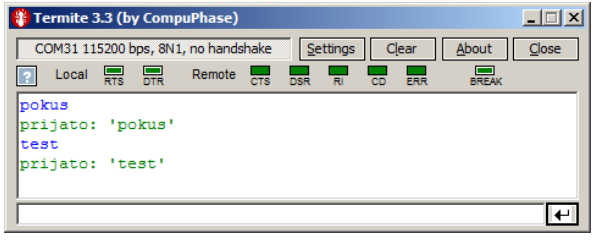
Kruhový buffer s DMA (4/4)

- Do funkce **uart_process_command()** je pro otestování vhodné doplnit zpětný výpis přijatého příkazu:

```
static void uart_process_command(char *cmd)
{
    printf("prijato: '%s'\n", cmd);
}
```

- Aby fungoval výpis pomocí **printf()**, je třeba inkludovat **<stdio.h>** a vytvořit systémovou funkci **_write()**:

```
int _write(int file, char const *buf, int n)
{
    /* stdout redirection to UART3 */
    HAL_UART_Transmit(&huart3, (uint8_t*)(buf), n, HAL_MAX_DELAY);
    return n;
}
```



Obsluha textového protokolu (1/2)

- Implementuje příkazy HELLO, LED1, LED2, LED3 a STATUS.
  - Příkaz HELLO vypíše uvítací text.
  - Příkazy LED1 a LED2 rozsvítí nebo zhasnou jednotlivé LED kontrolky podle údaje ON nebo OFF.
  - Příkaz STATUS vrátí stav obou LED.
- Použijte parsování příkazů pomocí funkce **strtok()** http://www.cplusplus.com/reference/cstring/strtok/, oddělovačem bude znak mezery, inkludujte hlavičkový soubor **<string.h>**. Jednotlivé příkazy rozlišujte se zanedbáním velikosti znaků pomocí funkce **strcasecmp()** http://www.cplusplus.com/reference/cstring/strcmp/.
- Využijte funkcí **HAL_GPIO_WritePin()**, **HAL_GPIO_ReadPin()** a formátovaný výpis. Opakovaná volání **strtok()** s prvním parametrem **NULL** vrací další tokeny ve formě C řetězců.

Obsluha textového protokolu (2/2)

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

```
char *token;
token = strtok(cmd, " ");

if (strcasecmp(token, "HELLO") == 0) {
    printf("Komunikace OK\n");
}
else if (strcasecmp(token, "LED1") == 0) {
    token = strtok(NULL, " ");
    if (strcasecmp(token, "ON") == 0) HAL_GPIO_WritePin(LD1_GPIO_Port, LD1_Pin, GPIO_PIN_SET);
    else if (strcasecmp(token, "OFF") == 0) HAL_GPIO_WritePin(LD1_GPIO_Port, LD1_Pin, GPIO_PIN_RESET);
    printf("OK\n");
}
else if (strcasecmp(token, "LED2") == 0) {
    // ...
}
// ...
else if (strcasecmp(token, "STATUS") == 0) {
    bool ld1 = HAL_GPIO_ReadPin(LD1_GPIO_Port, LD1_Pin);
    bool ld2 = HAL_GPIO_ReadPin(LD2_GPIO_Port, LD2_Pin);
    bool ld3 = HAL_GPIO_ReadPin(LD3_GPIO_Port, LD3_Pin);

    printf("Stav LED1=%s, LED2=%s, LED3=%s\n", ld1?"on":"off", ld2?"on":"off", ld3?"on":"off");
}
else {
    printf("Neznamy prikaz\n");
}
```

FreeRTOS, Ethernet, ADC
STM32F429

Systém FreeRTOS (1/2)

- Budeme využívat knihovnu lwIP pod systémem FreeRTOS. V CubeMX je třeba povolit FreeRTOS (CMSIS_V1) a podle doporučení přepnout časovou základnu (SYS / Timebase Source = TIM1).
- Vzhledem k rozsahu projektu je potřeba zvětšit velikost haldy FreeRTOS (FREERTOS / Config Parameters / Memory management settings / TOTAL_HEAL_SIZE = 32768).
- Veškeré úlohy pro Ethernet budou vytvářeny dynamicky, v konfiguraci FreeRTOS pod CubeMX je však potřeba vytvořit úlohu pro obsluhu USART z předchozích kroků, pokud chceme zachovat její funkci.

Tasks

Task Name	Priority	Stack...	Entry Function	Code Gen...	Parame...	Allocation	Buffer Name	Control Blo...
defaultTask	osPriorityNormal	128	StartDefaultTask	Default	NULL	Dynamic	NULL	NULL
SerialTask	osPriorityIdle	128	StartSerialTask	Default	NULL	Dynamic	NULL	NULL

- Úloha bude obsahovat inicializaci DMA přenosu a původní nekonečnou smyčku z main().

Systém FreeRTOS (2/2)

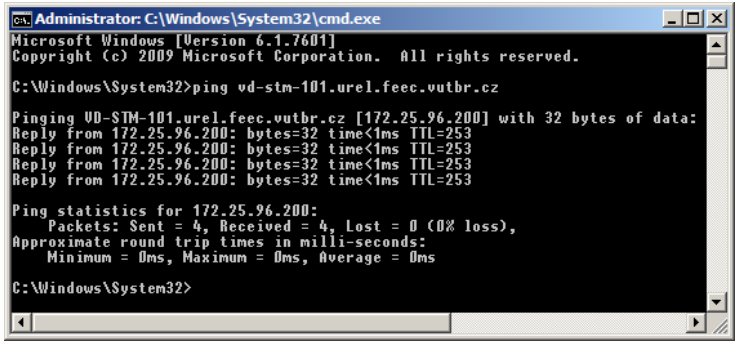

```
void StartSerialTask(void const *argument)
{
    /* USER CODE BEGIN StartSerialTask */
    HAL_UART_Receive_DMA(&huart3, uart_rx_buf, RX_BUFFER_LEN);

    /* Infinite loop */
    for (;;) {
        while (uart_rx_read_ptr != uart_rx_write_ptr) {
            uint8_t b = uart_rx_buf[uart_rx_read_ptr];
            if (++uart_rx_read_ptr >= RX_BUFFER_LEN)
                uart_rx_read_ptr = 0; // increase read pointer

            uart_byte_available(b); // process received byte
        }
        osDelay(1);
    }
    /* USER CODE END StartSerialTask */
}
```

Konfigurace Ethernetu

- Výchozí nastavení ethernetu je správné s výjimkou MAC adresy, ta musí být pro každou vývojovou desku jedinečná a zaregistrovaná v síti FEKT. Zvolte v ETH / Parameter Settings / Ethernet MAC Address adresu **00:80:E1:FE:EC:nn**, kde **nn** je číslo vašeho PC. Např. pro PC-071 tedy bude adresa 00:80:E1:FE:EC:71. Tomu odpovídá DNS adresa VD-STM-071.urel.feec.vutbr.cz.
- Protože je obsluha přerušení ETH periferie zajišťována přes volání FreeRTOS, musí být preemptivní priorita Ethernetu nižší než systémových volání RTOS (MAX_SYSCALL_INTERRUPT_PRIO, defaultně 5). V periferii NVIC tedy musíme nastavit prioritu Ethernet global interrupt na číslo numericky vyšší než 5 (např. 7).
- Aktivujte lwIP (Enabled), vygenerujte kód, přeložte a spusťte.

Otestování pomocí ping

- Po připojení do sítě se rozsvítí LED kontrolky na ethernetovém konektoru, ověřte funkčnost pomocí příkazu ping na adresu VD-STM-0**nn**.urel.feec.vutbr.cz.
- Získanou IP adresu lze ověřit pauznutím běhu programu a přidáním symbolu gnetif do okna Expressions. Po rozbalení gnetif / ip_addr / addr byste měli najít nenulovou IP adresu přiřazenou DHCP serverem v decimálním vyjádření.



Expression	Type	Value
gnetif	struct netif	{...}
next	struct netif *	0x0
ip_addr	ip_addr_t	{...}
addr	u32_t	3361741228
netmask	ip_addr_t	{...}
gw	ip_addr_t	{...}
input	netif_input_fn	0x8008bc1 <tcpip_input>
output	netif_output_fn	0x8012ed5 <etharp_output>
linkoutput	netif_linkoutput_fn	0x8005741 <low_level_output>

HTTP server (1/2)

- V konfiguraci lwIP aktivujte HTTP server (LWIP / HTTPD / LWIP_HTTPD = Enabled).
- Tento poměrně komplexní HTTP server je postavený na raw API. Webové stránky v nejjednodušším případě čte z paměti programu, kam jsou vloženy pomocí souboru fsdata_custom.c.
- Soubor fsdata_custom.c se vytváří pomocí programu makefsdata.exe (zdrojový kód je součástí balíku lwIP). Ten zkompilujte HTML soubory a obrázky ve složce Fs do souboru fsdata.c.
- Pro jednoduchost využijeme výchozí fsdata.c, který můžeme najít v repozitáři LwIP ve složce %USERPROFILE%\STM32Cube\Repository\STM32Cube_FW_F4_V1.25.1\Middlewares\Third_Party\LwIP\src\apps\http. Tento soubor přejmenujte na fsdata_custom.c, vložte ho mezi **hlavičkové** soubory projektu (Core/**Inc**) a vyřaďte z kompilovaných souborů (pravé tlačítko pro kontextové menu / Resource Configurations / Exclude from Build).
  - Tento postup řeší situaci, kdy soubor chceme mít přístupný v IDE, ale nekompilujeme ho samostatně – je totiž pomocí #include vložený do souboru Middlewares/Third_Party/LwIP/src/apps/http/fs.c. Musí tedy být v prohledávané cestě, aby fungoval #include (proto Core/Inc), a zároveň nesmí být samostatně kompilován (jinak by symboly v něm uvedené byly definované vícenásobně).

HTTP server (2/2)

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

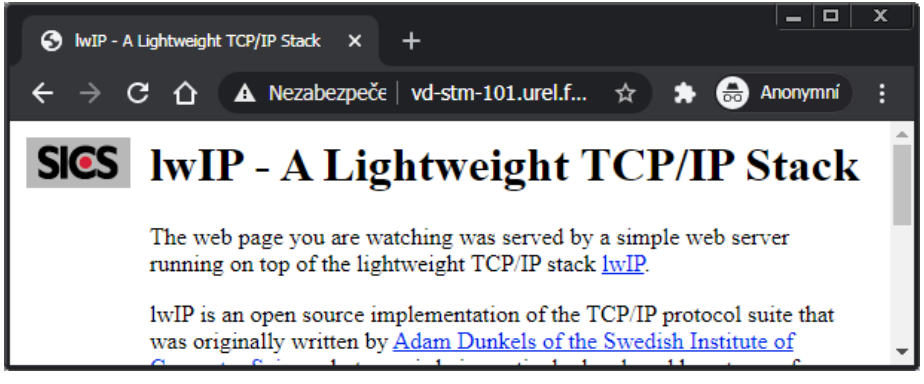
- V souboru main.c includujte příslušný hlavičkový soubor:

```
#include "lwip/apps/httpd.h"
```

- A ve StartDefaultTasku inicializujte HTTP server:

```
/* Initialize HTTP server */
httpd_init();
```

- Ověřte funkci pomocí webového prohlížeče na adrese http://VD-STM-0**nn**.urel.feec.vutbr.cz



Backup: Exponenciální kumulace pro ADC

- pro odstranění šumu a rušení ze signálu, typicky z AD převodníku
- význam předchozích repetic tím menší, čím jsou starší (postupné "zapomínání" starších hodnot)
- dlouhá kumulace umožňuje získat další bity z měření

```
#define apply_Q(x) ((x) >> 6)
static volatile uint32_t value_avg = 0;

/* HAL_ADC_ConvCpltCallback() */
value_avg -= apply_Q(value_avg);
value_avg += HAL_ADC_GetValue(hadc);

/* hlavni program */
printf("Hodnota=%u", apply_Q(value_avg));
```

$q = 1–2^{-6} = 0{,}984375$